Kauno technologijos universitetas

Informatikos fakultetas

Netiesinių lygčių sprendimas

P170B115 Skaitiniai metodai ir algoritmai

Pirmas laboratorinis darbas

Projekto autorius
Gustas Klevinskas

Akademinė grupė
IFF-8/7

Vadovas
Andrius Kriščiūnas

Kaunas, 2020

Turinys

Įvadas

Užduoties variantas – 15.

1 užduotis

Reikia daugianario, pateikto pirmoje formulėje, ir funkcijos, pateiktos antroje formulėje, sprendimus. Naudojami metodai: stygų, Niutono (liestinių), skenavimo su mažėjančiu žingsniu.

$$f(x) = 2.19x^4 - 5.17x^3 - 7.17x^2 + 15.14x + 1.21 \tag{1}$$

$$g(x) = e^{-\left(\frac{x}{2}\right)^2} sin(2x);\ -6 \le x \le 6 \tag{2}$$

2 užduotis

Vertikaliai į viršų iššauto objekto greitis užrašomas dėsniu trečioje formulėje.

$$v(t) = v_0 e^{-\frac{ct}{m}} + \frac{mg}{c}\left(e^{-\frac{ct}{m}} - 1\right) \tag{3}$$

Čia: $g = 9.8\ m/s^2$, pradinis greitis v_0, objekto masė m. Koks pasipriešinimo koeficientas c veikia objektą, jei žinoma, kad po t_1 laiko nuo iššovimo jo greitis lygus v_1?

1 lentelė. 15 varianto duomenys

v_0, m/s	m, kg	t_1, s	v_1, m/s
50	2	3	14

Pirma užduotis

Intervalai

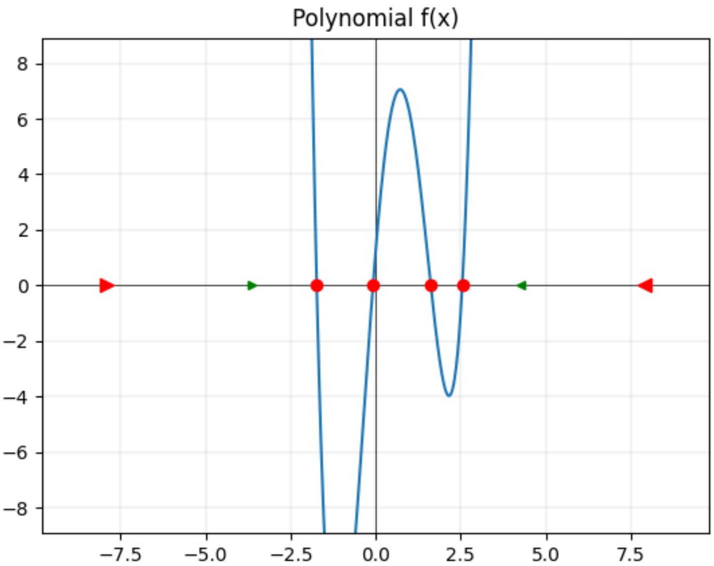
Daugianario grubus intervalas: $|x| < 7.913$.

Daugianario tikslesnis intervalas: $-3.629 \leq x \leq 4.274$.

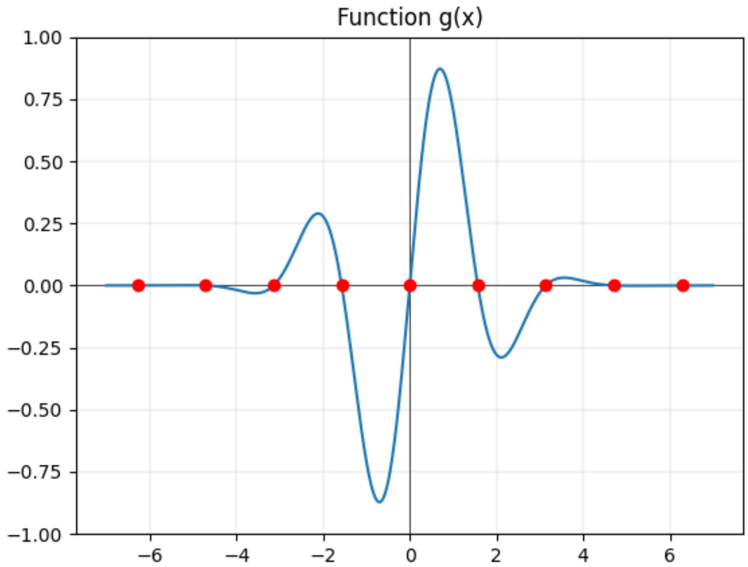
Grafikai

Pirmame paveiksle pateiktas $f(x)$ grafikas. Raudonais trikampiais pažymėtas grubus intervalas, mažesniais žaliais – tikslesnis.

1 ir 2 paveiksluose raudonais taškais pažymėtos rastos šaknys Niutono metodu.



1 pav. f(x) grafikas



2 pav. g(x) grafikas

Visų trijų metodų pabaigos sąlyga yra absoliutus įvertis. Stygų ir Niutono metoduose tai yra $|f(x)| < \varepsilon$, kur ε – norimas tikslumas. Skenavimo su mažėjančiu žingsniu metode tikrinama, ar x intervalo ilgis yra mažesnis už norimą tikslumą ($\Delta x < \varepsilon$).

Rezultatai

Antroje lentelėje pavaizduotos apskaičiuotos šaknys, apskaičiuotos stygų, Niutono ir skenavimo metodais.

2 lentelė. Skaičiavimų rezultatai

Metodas	Pradinis artinys arba intervalas	Apskaičiuota šaknis	`np.roots()` šaknis	Iter. sk.
		$f(x)$		
Stygų	[-1.7393044725045463, -1.6393044725045463]	-1.7310729963868994	-1.7310729963868976	7
Niutono	-1.7393044725045463	-1.7310729963868998		3
Skenavimo	[-1.7393044725045463, -1.6393044725045463]	-1.7310729963868967		12
Stygų	[-0.13930447250454492, -0.03930447250454483]	-0.07725674294712696	-0.07725674294712831	8
Niutono	-0.13930447250454492	-0.07725674294712834		4
Skenavimo	[-0.13930447250454492, -0.03930447250454483]	-0.07725674294713034		13
Stygų	[1.5606955274954553, 1.6606955274954553]	1.6219937959711235	1.6219937959711241	7
Niutono	1.5606955274954553	1.621993795971124		3
Skenavimo	[1.5606955274954553, 1.6606955274954553]	1.6219937959710975		12
Stygų	[2.460695527495456, 2.560695527495456]	2.54706653697021	2.547066536970212	9
Niutono	2.460695527495456	2.547066536970211		5
Skenavimo	[2.460695527495456, 2.560695527495456]	2.547066536970209		12
		$g(x)$		
Stygų	[-6.31, -6.21]	-6.2831853078075035	-6.283185307179586	11
Niutono	-6.31	-6.283185307179586		4
Skenavimo	[-6.31, -6.21]	-6.283185307179586		13
Stygų	[-4.81, -4.71]	-4.712388980385325	-4.71238898038469	5
Niutono	-4.81	-4.71238898038469		5
Skenavimo	[-4.81, -4.71]	-4.712388980384693		13
Stygų	[-3.21, -3.11]	-3.141592653589887	-3.141592653589793	9
Niutono	-3.21	-3.1415926535897922		4
Skenavimo	[-3.21, -3.11]	-3.1415926535897647		12
Stygų	[-1.61, -1.51]	-1.5707963267949194	-1.5707963267948966	9
Niutono	-1.61	-1.5707963267948966		4

Skenavimo	[-1.61, -1.51]	-1.570796326794864		12
Stygų	[-0.01, 0.09]	2.42669194222874e-17	0	3
Niutono	-0.01	-1.1303866439345e-17		2
Skenavimo	[-0.01, 0.09]	3.23163593897335e-15		13
Stygų	[1.49, 1.59]	1.5707963267949023	1.5707963267948966	10
Niutono	1.49	1.5707963267948966		4
Skenavimo	[1.49, 1.59]	1.5707963267949308		12
Stygų	[3.09, 3.19]	3.141592653590109	3.141592653589793	10
Niutono	3.09	3.141592653589793		4
Skenavimo	[3.09, 3.19]	3.141592653589842		12
Stygų	[4.69, 4.79]	4.71238898038869	4.71238898038469	8
Niutono	4.69	4.71238898038469		4
Skenavimo	[4.69, 4.79]	4.712388980384686		13
Stygų	[6.19, 6.29]	6.283185307550204	6.283185307179586	12
Niutono	6.19	6.283185307117488		4
Skenavimo	[6.19, 6.29]	6.283185307179583		13

Išvados

Mažiausiai iteracijų gaunama naudojant Niutono metodą. Tačiau norint naudoti jį reikia žinoti funkcijos išvestinę. Ją lengva rasti, jei funkcija yra polinomas, tačiau $g(x)$ funkcijos išvestinę reikėjo rasti naudojant išorinius šaltinius (aš naudojau Wolfram Alpha).

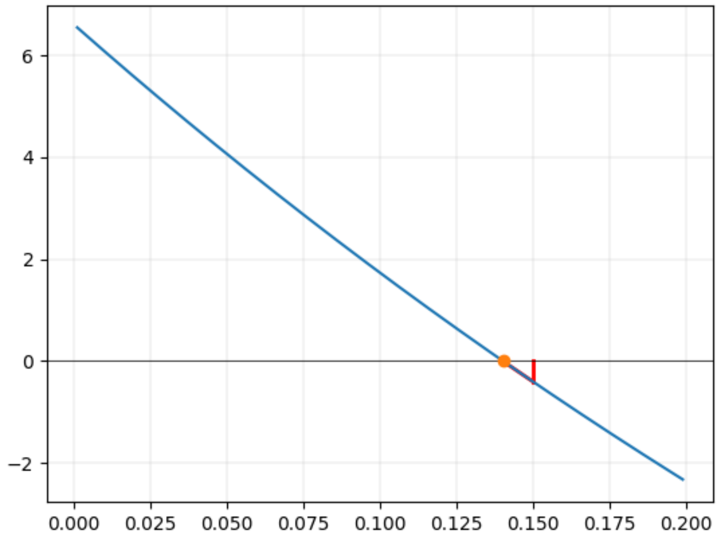
Antra užduotis

Antrą užduotį pasirinkau spręsti naudojant Niutono metodą. Pradinį žingsnį pasirinkau 0.15. Metodo pabaigos sąlyga $|f(x)| < 10^{-10}$.

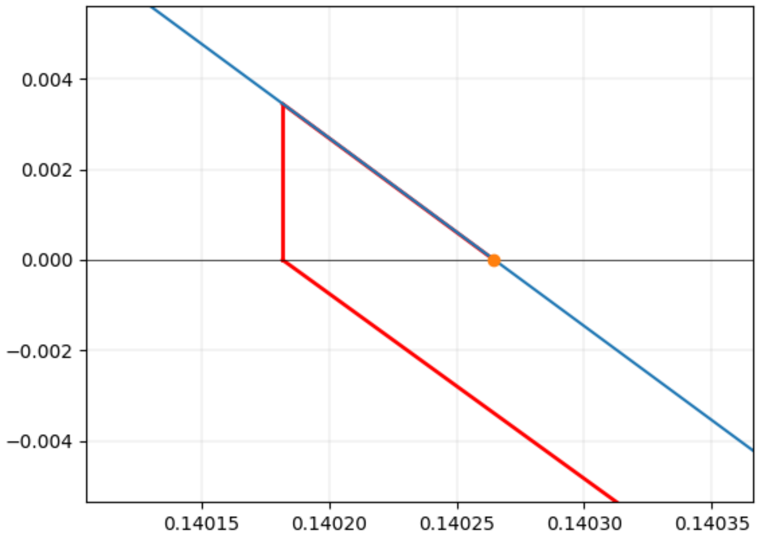
Šį metodą pasirinkau, nes funkcija yra gana artima tiesei. Kadangi Niutono metodas naudoja išvestines sekančios x reikšmės gavimui, jis šiuo atveju labai greitai randa šaknį. Ir šį metodą jau buvau apsirašęs sprendžiant pirmąją užduotį.

Gauti rezultatai:

Šaknis: 0.14026464103334876, gauta per 3 iteracijas.



3 pav. Lygties grafikas



4 pav. Priartintas lygties grafikas

Programinis kodas

L1_1.py

```
import matplotlib.pyplot as plt
import solve
from functions import *
from intervals import *


# ---- Plotting f(x) ---- #
plt.figure(0)
plt.title('Polynomial f(x)')

# Calculating x interval to plot
x_from = min(acc_lower(COEFF), -r(COEFF)) - 1
x_to = max(acc_upper(COEFF), r(COEFF)) + 1

# Plot setup
plt.ylim(x_from, x_to)
plt.axvline(linewidth=0.5, color='k')
plt.axhline(linewidth=0.5, color='k')
plt.grid(linewidth=0.2)

# Plotting rough interval estimate
plt.plot(-r(COEFF), 0, color='r', marker='>', markersize=7)
plt.plot(r(COEFF), 0, color='r', marker='<', markersize=7)

# Plotting accurate interval estimate
plt.plot(acc_lower(COEFF), 0, color='g', marker='>', markersize=5)
plt.plot(acc_upper(COEFF), 0, color='g', marker='<', markersize=5)

x1 = np.arange(x_from, x_to, 0.01)
y1 = f(x1)
plt.plot(x1, y1)

intervals = solve.get_all_intervals(f, acc_lower(COEFF), acc_upper(COEFF), 0.1)
for i, interval in enumerate(intervals):
    cor, it_cor = solve.chord(f, interval[0], interval[1])
    ntn, it_ntn = solve.newton(f, f_prime, interval[0])
    scn, it_scn = solve.scan(f, interval[0], interval[1])

    print('----- f(x) root #{} -----'.format(i + 1))
    print('Chord  = ', cor, ' iterations = ', it_cor)
    print('Newton = ', ntn, ' iterations = ', it_ntn)
    print('Scan   = ', scn, ' iterations = ', it_scn)
    print()

    plt.plot(ntn, 0, 'ro')


# ---- Plotting g(x) ---- #
plt.figure(1)
plt.title('Function g(x)')

x_from = -6 - 1
x_to = 6 + 1

# Plot setup
plt.ylim(-1, 1)
plt.axvline(linewidth=0.5, color='k')
plt.axhline(linewidth=0.5, color='k')
plt.grid(linewidth=0.2)

x1 = np.arange(x_from, x_to, 0.01)
y1 = g(x1)
plt.plot(x1, y1)
```

```
intervals = solve.get_all_intervals(g, x_from, x_to, 0.1)
for i, interval in enumerate(intervals):
    cor, it_cor = solve.chord(g, interval[0], interval[1])
    ntn, it_ntn = solve.newton(g, g_prime, interval[0])
    scn, it_scn = solve.scan(g, interval[0], interval[1])

    print('----- g(x) root #{} -----'.format(i + 1))
    print('Chord  = ', cor, ' iterations = ', it_cor)
    print('Newton = ', ntn, ' iterations = ', it_ntn)
    print('Scan   = ', scn, ' iterations = ', it_scn)
    print()

    plt.plot(ntn, 0, 'ro')


plt.show()
```

functions.py

```
import numpy as np

COEFF = [2.19, -5.17, -7.17, 15.14, 1.21]


def _derivative(coefficients):
    _coeff = coefficients[:-1]
    exp = len(coefficients) - 1

    for i in range(len(_coeff)):
        _coeff[i] *= exp
        exp -= 1

    return _coeff


def _f(x, coeff):
    y = 0.0
    i = len(coeff) - 1

    for coefficient in coeff:
        y += coefficient * x ** i
        i -= 1

    return y


def f(x):
    return _f(x, COEFF)


def f_prime(x):
    return _f(x, _derivative(COEFF))


def g(x):
    return np.exp(-((x / 2) ** 2)) * np.sin(2 * x)


def g_prime(x):
    return 0.5 * np.exp(-((x / 2) ** 2)) * (4 * np.cos(2 * x) - x * np.sin(2 * x))
```

intervals.py

```
import numpy as np


def _b(coefficients):
    only_negatives = []

    for i in coefficients[1:]:
        if i < 0:
            only_negatives.append(i)

    if len(only_negatives) == 0:
        return 0

    return max(list(map(lambda num: abs(num), only_negatives)))


def _k(coefficients):
    n = len(coefficients) - 1
    max_index = -1

    for i, v in enumerate(coefficients[1:]):
        if v < 0:
            max_index = n - (i + 1)
            break

    if max_index == -1:
        return 0

    return n - max_index


def r(coefficients):
    coefficients = coefficients[:]

    if coefficients[0] < 0:
        coefficients = np.negative(coefficients)

    return 1 + max(coefficients[1:]) / coefficients[0]


def r_pos(coefficients):
    __b = _b(coefficients)
    __k = _k(coefficients)

    if __b == 0 or __k == 0:
        return 0

    return 1 + (__b / coefficients[0]) ** (1. / __k)


def r_neg(coefficients):
    _coefficients = coefficients[:]
    n = len(_coefficients) - 1

    for i in range(n + 1):
        if ((n - i) % 2) != 0:
            _coefficients[i] = -_coefficients[i]

    if _coefficients[0] < 0:
        _coefficients = np.negative(_coefficients)

    __b = _b(_coefficients)
    __k = _k(_coefficients)

    if __b == 0 or __k == 0:
        return 0
```

```
    return 1 + (__b / _coefficients[0]) ** (1. / __k)


def acc_lower(coefficients):
    return -min(r(coefficients), r_neg(coefficients))


def acc_upper(coefficients):
    return min(r(coefficients), r_pos(coefficients))
```

solve.py

```
import numpy as np

ACCURACY = 1e-13
BETA = 1


def get_interval(f, x_from, x_to, step):
    start = x_from

    for x in np.arange(x_from, x_to, step):
        if np.sign(f(start)) != np.sign(f(x)):
            return start, x

        start = x

    return None, None


def get_all_intervals(f, x_from, x_to, step):
    x_from -= step * 1.1
    x_to -= step * 1.1

    start, end = get_interval(f, x_from, x_to, step)
    intervals = []

    while start is not None and end is not None:
        intervals.append([start, end])
        start, end = get_interval(f, end, x_to, step)

    return intervals


def chord(f, x_from, x_to):
    iterations = 1

    k = abs(f(x_from) / f(x_to))
    x_mid = (x_from + k * x_to) / (1 + k)

    while abs(f(x_mid)) > ACCURACY:
        iterations += 1

        if np.sign(f(x_mid)) == np.sign(f(x_to)):
            x_to = x_mid
        else:
            x_from = x_mid

        k = abs(f(x_from) / f(x_to))
        x_mid = (x_from + k * x_to) / (1 + k)

    return x_mid, iterations


def newton(f, f_prime, x0):
    iterations = 0

    while abs(f(x0)) > ACCURACY:
        x0 = x0 - BETA * (f(x0) / f_prime(x0))
        iterations += 1

    return x0, iterations


def scan(f, x_from, x_to):
    iterations = 0
    step = (x_to - x_from) / 10
    start = x_from
```

```python
while abs(x_to - x_from) > ACCURACY:
    for x in np.arange(x_from, x_to, step):
        if np.sign(f(start)) != np.sign(f(x)):
            x_from = start
            x_to = x
            break

        start = x

    step /= 10
    iterations += 1

return (x_from + x_to) / 2, iterations
```

L1_2.py

```
import matplotlib.pyplot as plt
import numpy as np
import sympy

v0 = 50.0
m = 2.0
t1 = 3.0
v1 = 14.0
BETA = 1


def newton(x0):
    iterations = 0

    plt.plot([x0, x0], [0, fun(x0)], linewidth=2, c='r')

    while abs(fun(x0)) > 1e-10:
        x0old = x0
        fold = fun(x0)
        x0 = x0 - BETA * (fun(x0) / fun_prime(x0))
        plt.plot([x0old, x0], [fold, 0], linewidth=2, c='r')
        plt.plot([x0, x0], [0, fun(x0)], linewidth=2, c='r')
        iterations += 1

    return x0, iterations


c = sympy.Symbol('c')
f = v0 * sympy.exp(-(c * t1) / m) + (m * 9.8 * (sympy.exp(-(c * t1) / m) - 1)) / c - v1
f_prime = f.diff(c)

fun = sympy.lambdify(c, f)
fun_prime = sympy.lambdify(c, f_prime)

root, iterations = newton(0.15)
print('Root = ', root)
print('Iterations = ', iterations)

x1 = np.arange(0, 0.2, 0.001)
y1 = fun(x1)

plt.axhline(linewidth=0.5, color='k')
plt.grid(linewidth=0.2)
plt.plot(x1, y1)
plt.plot(root, 0, 'o')

plt.show()
```